Nečekaní zabijáci výkonu databázových dotazů

RNDr. David Gešvindr
MVP: Data Platform | MCSE: Data Platform | MCT
david@wug.cz
@gesvindr

Nevhodný návrh dotazu

- Efektivní dotaz načítá z databáze jen **nutné minimum informací**, které dostačuje aplikaci pro daný účel
  - Např.: Pro vypsání seznamu objednávek nemusím načítat 50 dalších sloupců v tabulce objednávek, když nejsou zobrazeny v UI

- Je důležité:
  - Vracet jen sloupce, co skutečně využijeme (pozor na SELECT * FROM tabulka)
  - Filtrovat a stránkovat záznamy na serveru
  - Zbytečně neřadit záznamy, pokud to opravdu nepotřebujeme

Jak zjistit složitost dotazu

- **Exekuční plán** popisuje, jaké fyzické operace SQL Server musel realizovat pro načtení dat
  - Je vyčíslena cena dotazu
  - **Pozor – Tato cena je vypočtena na základě odhadovaného exekučního plánu a nemusí odpovídat skutečnosti!**
  - Aktuální SSMS umí zobrazit skutečné složitosti vybraných operací (SQL Server 2016 a SQL Server 2014 SP2; SQL Server 2016 SP1 i wait statistiky na úrovni dotazu)
- **SET STATISTICS IO ON**
  - SQL Server ukáže počet I/O operací spojených s exekucí dotazu
- **SET STATISTICS TIME ON**
  - SQL Server ukáže pro každý příkaz v dávce dobu kompilace a exekuce

Zdroj: [1] https://blogs.msdn.microsoft.com/sql_server_team/extended-per-operator-level-performance-stats-for-query-processing/
[2] https://blogs.msdn.microsoft.com/sql_server_team/new-showplan-xml-properties-in-ssms-october-release/

Iterativní zpracování

- SQL Server je optimalizován pro množinové zpracování dat
- Databázová tabulka je množina řádků a databázový dotaz je popis její transformace
  - Deklarativní zápis, co bude výsledkem dotazu, nikoliv jak k němu dojít
- Pokud budeme data zpracovávat řádek po řádku, dojde k propastnému snížení výkonu
  - Kurzory
  - Cykly zpracovávající data po individuálních řádcích
  - Funkce

Skalární funkce

- Skalární funkce vrací jednu hodnotu a na vstupu má 0..n parametrů
- Skalární funkce bývají častým výkonnostním problémem
  - Nejsou součástí exekučního plánu dotazu
  - Volají se vždy nezávisle pro každou sadu vstupních parametrů
  - Nejsou oceněny (SQL Server neví, jak je exekuce funkce drahá)
  - Blokují paralelní zpracování

- Řešení? Nepoužívat skalární funkce nebo pořídit SQL Server 2019 ☺

Scalar UDF inlining

- SQL Server 2019 analyzuje skalární funkci a pokud je to možné její tělo automaticky vloží do samotného dotazu.
  - Zahrnuje komplexní transformace zdrojového kódu funkce (IF → CASE)
  - Odpadnou nevýhody spojené s exekucí skalární funkce

- Požadavky kladené na skalární funkci:
  - Funkce používá DECLARE, SET, SELECT, IF/ELSE, RETURN, další UDF
  - Nesmí záviset na čase, nesmí používat EXECUTE AS, nesmí používat TV/TVP

Zdroj: https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining

Nevhodně napsané podmínky

- Pokud je v podmínce nevhodně zapsaný výraz či skalární funkce, dojde k nutnosti vyhodnotit hodnotu výrazu pro každý řádek tabulky

- Výraz v podmínce WHERE by měl být, je-li to možné, tzv. „SARGABLE“ [1]
  - Podmínku lze rozhodnout na úrovni operátoru přistupujícího k datům podle hodnot přečtených v tabulce či indexu
  - To umožňuje efektivně číst pouze podmnožinu řádků použitím operace Index Seek

- Nefiltrovat dle hodnoty výrazu, kde dochází k transformaci hodnot sloupců
  - Je nutné používat skalární funkce ve výrazu podmínky tak, aby neměly jako vstupní parametry hodnoty sloupců

Zdroj: [1] https://cs.stanford.edu/people/chrismre/cs345/rl/selinger.pdf

Implicitní konverze v podmínce

- Při porovnání různých datových typů v podmínce dotazu musí dojít k implicitní konverzi na stejný datový typ
- Pokud směr implicitní konverze vede ke konverzi sloupce v tabulce místo parametru, musí se konverze vyhodnotit pro všechny hodnoty v tabulce
  - Směry implicitní konverze jsou popsány v dokumentaci [1]
- Implicitní konverze navíc vede k nemožnosti odhadnout kardinalitu

Zdroj: [1] https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-type-precedence-transact-sql?view=sql-server-ver15

Multi-statement Table-Valued funkce

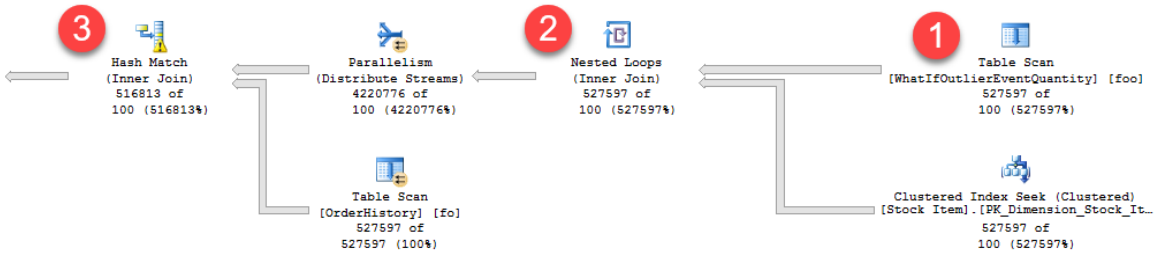
- Table-Valued funkce vrací množinu řádků a má 0..n vstupních parametrů (bývá označována jako „parametrizovaný pohled“)

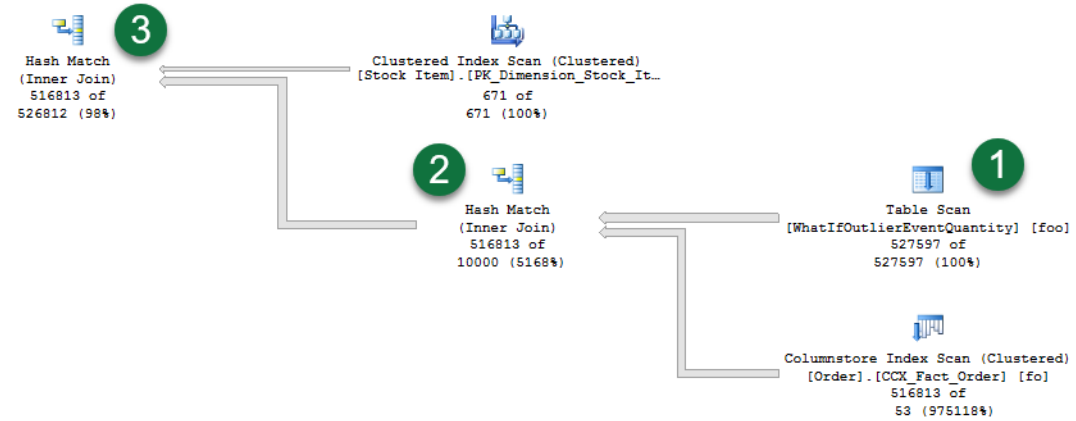
- **Inline Table-Valued funkce** svůj kód vkládají jako součást volajícího dotazu
  - Generuje se jeden exekuční plán a data se zpracovávají množinově

- **Multi-statement Table-Valued funkce** se nemohou vložit do volajícího dotazu
  - Mají nezávislou exekuci pro každé volání
  - Na Microsoft SQL Serveru 2016 a starších mají pevný odhad kardinality

Interleaved Execution

- SQL Server 2017 umí přerušit sestavení exekučního plánu, vyhodnotit multi-statement table-valued funkci a použít pro sestavení exekučního plánu skutečný počet řádků
  - Funkce se aktivuje nastavením kompatibility levelu databáze na verzi SQL Server 2017 a novější
  - Dřívější verze SQL Serveru pracovaly s pevným odhadem 100 řádků





Zdroj: https://docs.microsoft.com/en-us/sql/relational-databases/performance/adaptive-query-processing

Table Variables / Temporary Tables

Table Variables

- Platnost v rámci dávky
- Ukládá se do tempdb
- Méně rekompilací uložených procedur
- Neprovede se na nich ROLLBACK
- Má pevný odhad počtu řádků 1
  - S výjimkou SQL Serveru 2019

- Jak je dostat jen do paměti?

Session Temporary Table

- Platnost v rámci session
- Ukládá se do tempdb
- Více rekompilací uložených procedur
- Má správný odhad počtu řádků

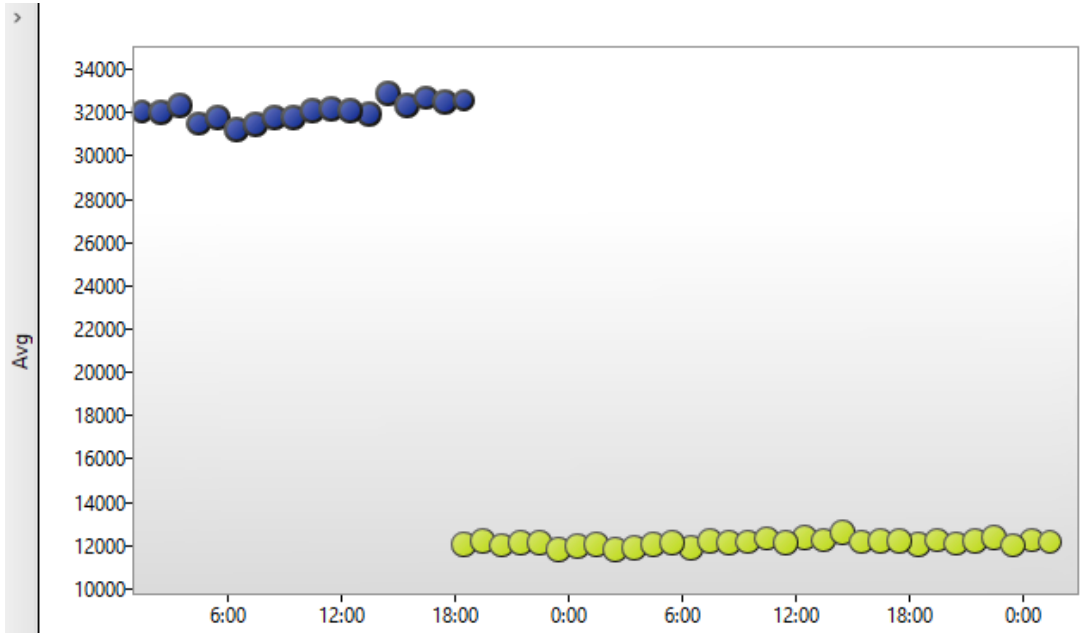
Table Variable Deferred Compilation

- Table Variables mají pevný odhad kardinality **1 řádek**
  - Exekuční plány využívající tabulkové proměnné mívají často výkonnostní problémy, protože v těchto proměnných může být velké množství řádků

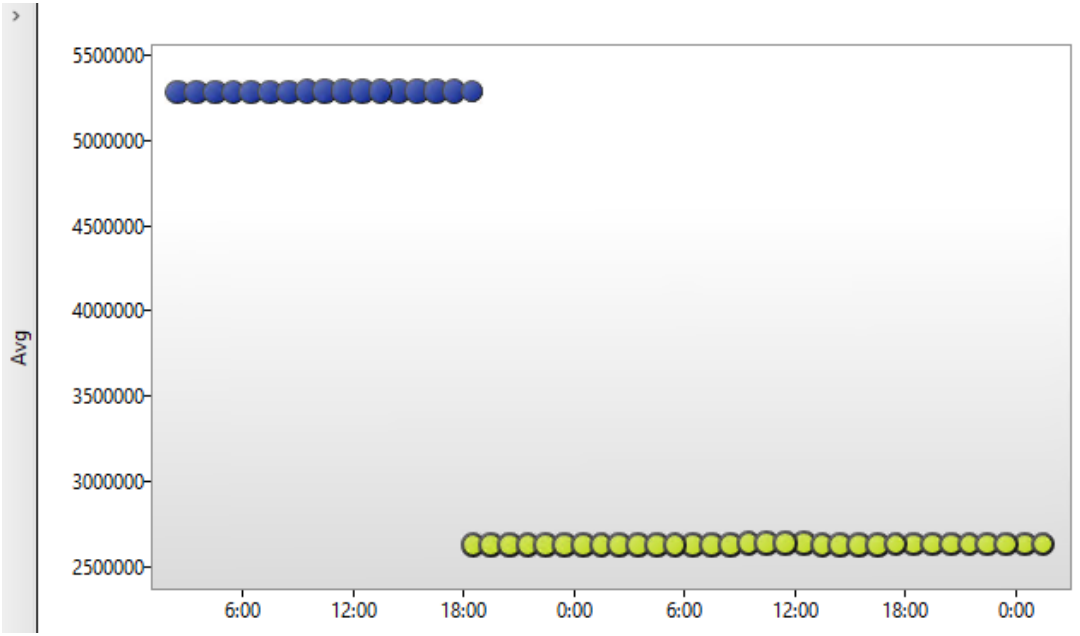
- V kompatibility levelu 150 je pozdržena kompilace dotazů pracujících s tabulkovou proměnnou, **dokud do ní nejsou načtena data**
  - Díky tomu je známý skutečný počet řádků v této proměnné a to je zohledněno při sestavení exekučních plánů dotazů, které ji používají
  - Vygenerovaný exekuční plán je nacachován, pokud se tedy výrazně mění počet řádků v proměnné mezi exekucemi, mohou nastat problémy s parameter sniffingem

Dopad kompatibility levelu 150

- Zkušenost: Nejdražší dotazy **zatížené zmíněnými problémy** konzumují 2-3x méně zdrojů jen díky změně nastavení databáze



CPU Time



Logical Reads

Zastaralé statistiky

- SQL Server potřebuje odhadnout, kolik řádků bude zpracovávaných jednotlivými operátory v exekučním plánu
  - Odhad kardinality
  - Různé varianty exekučního plánu se vyplatí pro různý počet řádků

- Statistiky se neaktualizují spolu s daty
  - AUTO UPDATE STATISTICS
  - Pravidelná údržba

Parameter sniffing

- Když SQL Server kompiluje exekuční plán dotazu s parametry, optimalizuje dotaz pro hodnoty parametrů při prvním volání
- Tento plán ale nemusí být efektivní pro ostatní hodnoty parametrů
- Řešení:
  - Rekompilace dotazu vždy před spuštěním OPTION (RECOMPILE)
    - Rekompilace celé procedury (pozor) před spuštěním WITH RECOMPILE
  - Instrumentace optimalizátoru, která hodnota parametru je nejlepší pro sestavení plánu:
    OPTION (OPTIMIZE FOR (@CustomerID=800));
    OPTION (OPTIMIZE FOR UNKNOWN); (stejný efekt jako použití proměnné)
  - Rozdělení procedury na víc procedur

Zámky

- SQL Server využívá mechanizmus zámků pro řízení souběžného přístupu k datům (pessimistic concurrency)
  - S výjimkou verzování (Snapshot Isolation level, In-Memory OLTP)
- Důsledkem použití zámků je blokování
  - Jedna dlouho trvající transakce může zablokovat přístup k většině systému
- Použité izolační úrovně určují rozsah blokování a také možné problémy při souběžném zpracování
- Bližší informace o fungování zámků:
  - https://www.wug.cz/zaznamy/331-SQL-Server-Bootcamp-2016-Transakce-zamky-a-izolacni-urovne-v-SQL-Serveru

Eliminace zbytečných poddotazů

- Poddotazy nejsou obecně výkonnostní problém
  - Např.: Osobně preferuji EXISTS před JOIN, pokud mi jde pouze o filtraci
  - Pokud dotaz vrací stejný výsledek, často vede ke stejnému exekučnímu plánu

- Pokud potřebujeme v poddotazu počítat agregace pracující s množinou výsledku **může** použití **OVER (Window Functions)** přinést vyšší výkon
  - Window funkce realizuje výpočet na podmnožině řádků výsledků dotazu
  - Podmnožina je počítána podle pravidel pro každý řádek dotazu
  - Podmnožina je předána agregační, rank či offset funkci

Dotazy

Chcete si na toto téma společně povídat 4 dny? Přijďte na GOC 631

RNDr. David Gešvindr

MVP: Data Platform | MCSE: Data Platform | MCT

david@wug.cz

@gesvindr